A HEURISTIC APPROACH FOR IMPROVING SUB-OPTIMAL SEUQENCE OF A STATIC FLOWSHOP SCHEDULING PROBLEM

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
RAJIVA CHAWLA

to the
INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
JULY, 1979

CERTIFICATE

This is to certify that the present work on 'A Heuristic Approach for Improving Sub-Optimal Sequence of a Static Flow-Shop Scheduling Problem' by Rajiva Chawla has been carried out under our supervision and has not been submitted elsewhere for the award of a degree.

(S.K. Gupta)
Asst.Professor
Mathematics Dept.,
IIT Kanpur

(J.L. Batra)
Professor and Head
Industrial and Management Engg.
IIT Kanpur

ACKNOWLEDGEMENTS

With great pleasure, I express my indebtness to Dr. J.L. Batra and Dr. S.K. Gupta for their invaluable help and guidance, throughout the course of this work.

I am thankful to Mr. J.K. Misra for the flawless typing of this manuscript. Thanks are also due to Mr. Buddhi Ram for having cyclostyled this thesis.

Rajiva Chawla

CONTENTS

SYNOPSIS

The present work deals with static flow-shop scheduling problem. Many exact and heuristic procedures have been reported in the literature for this purpose considering minimization of makespan time as the optimization criterion. The exact procedures yield optimal solution but tend to be highly computationally inefficient for large sized problems involving more than two machines. The reported heuristic procedures, though computationally fairly efficient, do not guarantee the optimality of the solution.

Given a suboptimal solution obtained by using one of the known heuristics, one can possibly exploit the characteristics of the solution for developing heuristic rules which when used on this sub-optimal sequence would yield improved solution. In a broad sense, this is the basic idea used in the present work.

The given suboptimal solution would result in idle time for some jobs as well as the machines. The information about the idle time has been to an advantage for the development of proposed heuristic which is used for improving upon the suboptimal sequence. The concepts of Earliest and Latest Finish Gantt matrices, slack time and critical path have been introduced for the development of the

heuristic. The heuristic is iterative in nature and in each iteration it identifies the jobs which become the candidates for the cyclic pairwise exchange. The iterative procedure is continued till it fails to generate any further improved sequence.

Computer programmes for the Petrov's algorithm to obtain the initial sequence and for the proposed heuristic procedure have been developed. The performance of the proposed heuristic has been tested on a number of randomly generated problems of various sizes. It was observed that the heuristic improved the initial sequence for about 70 percent of the test problems, further on an average there was a reduction of about 8 percent in the makespan time for the problems in which the proposed heuristic was successful in generating the improved sequence.

CHAPTER I

INTRODUCTION

Scheduling is the allocation of resources over time to perform a collection of tasks. This rather general definition when conceived in the context of manufacturing, assumes the resources to be machines and the tasks to be jobs or products that have to be processed.

The term scheduling conveys two different meanings that are important to be mentioned here. First, scheduling is a decision-making function: it is the process of determining a schedule. Second, scheduling is a body of theory: it is a collection of principles, models, techniques and logical conclusions that provide insight into the scheduling function.

The other term that is used quite frequently in this context is 'sequencing'. Sequencing is concerned with the arrangements and permutations in which a set of jobs under consideration are performed on all machines. Scheduling is concerned with the specification of starting or completion times of jobs on all machines. Sequencing decisions focus on the arrangements of events, whereas scheduling decisions focus on the time of events.

The present work deals with the flow-shop scheduling problem. In a flow-shop, jobs are multistage in nature and the machines have serial configuration. A job is considered to be a collection of operations in which a special precedence structure applies. In particular each operation after the first has exactly one direct predecessor and each operation before the last has exactly one direct successor. This is depicted in Fig. 1.1 for a job j. The numbers 1,2,..., m, refer to the machines and each circle denotes an operation. Each

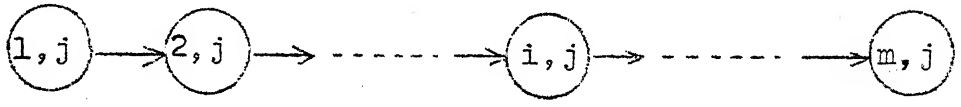


Fig. 1.1

operation requires a different machine. A flow-shop is characterized by unidirectional flow of work, or in other words, it is possible to number the machines so that if the i-th operation of any job j precedes its k-th operation then the machine required by the i-th operation has a lower number than the machine required by the k-th operation. The machines in a flow-shop are numbered 1,2,..., m and the operations of job j are correspondingly numbered (1,j), (2,j),..., (m,j). Each job can be treated as if it had exactly m operations, for in cases where fewer operations exist, the corresponding processing times can be taken to be zero.

Given a flow-shop environment with m machines and n jobs, the objective is to determine an optimal sequence considering some measure of effectiveness. The effectiveness of a sequence can be measured in terms of makespan time (total elapsed time), average completion time, due date performance, machine utilization, inventory of jobs in process, etc. Considerable research effort has been directed by many researchers towards the solution of above stated combinatorial optimization problem under a set of standard assumptions (1).

Most of the researchers have considered the minimization of makespan time as the criteria for generation of optimal sequence. For a 2-machine, n job situation Johnson has presented an algorithm for obtaining an optimal solution. As the number of machines increases beyond two, the use of operation research technique which can yield optimal solutions becames restrictive due to large computational requirements. Therefore, number of heuristic procedures have been suggested which though computationally efficient can only yield near optimal solutions to the problem. The near optimal sequence obtained by any one of the available heuristic will, in general, have some idle times associated with its operations. The immediate question which comes to ones mind is, can these idle-times be exploited for the

generation of a better sequence? The present work essentially concerns itself with the generation of better sequence through the use of idle-time information. The criteria of minimizing make-span time is used for the evaluation of the sequences. Further, the standard set of assumptions listed by Bakshi and Arora (1) are considered for the present work.

The general frame work of the proposed procedure is depicted in Fig. 1.2.

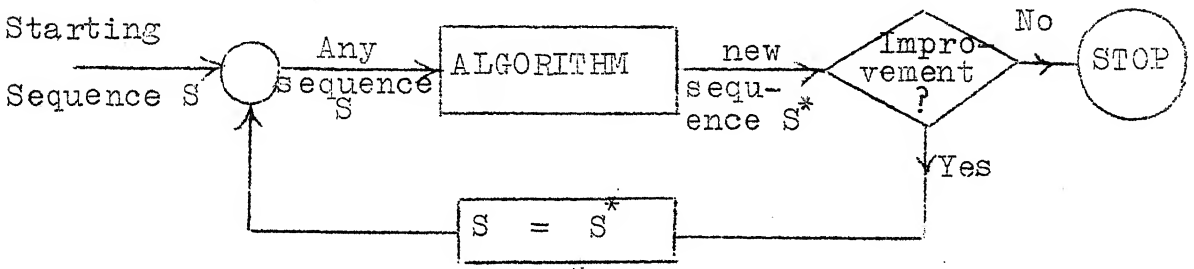


Fig. 1.2

This is a kind of feed-back system in which the initial sequence can be any sequence. The algorithm attempts to improve upon the given sequence S. Each application of the algorithm yields a new sequence. If the new sequence S^* happens to be better than the previous one, the algorithm is applied again on the new sequence. The procedure continues till no further improvement is possible.

It should be noted here that better the initial sequence, faster is the convergence towards final

improved sequence for the problem under consideration. Therefore, the initial sequence is generated by applying any one of the available heuristics instead of choosing it randomly.

The plan of the thesis is as follows. In Chapter II, a brief review of the literature on heuristic procedures for flow-shop scheduling problem is presented. Chapters III is devoted to the development of basic frame-work which is used for structuring the proposed heuristic algorithm presented in Chapter IV. The proposed algorithm has been tested on standard problems available in the literature and also on a large number of randomly generated problems of varied sizes. The results are presented in Chapter V.

CHAPTER II

LITERATURE SURVEY

In the area of flow-shop scheduling, considerable research work has been reported in the literature. Minimization of make-span time and average flow time are generally considered as the optimization criteria. A thorough survey of the literature indicates that the flow shop problems have been tackled using mathematical programming, complete enumeration, Monte Carlo sampling, decomposition and Heuristic approaches. Since the present work is primarily concerned with the development of heuristic approaches for flow-shop scheduling problems, only the literature relevant to this is reviewed in the following section.

2.1 Heuristic Approaches:

Heuristic models are developed utilizing logic and common sense derived from the observation of the behavior of solutions under varied conditions. Heuristic procedures never guarantee optimality. These are some sort of intuitive proposals that would give, in most of the cases, near optimal results. The effectiveness of a heuristic is judged by its easiness and closeness of its results to the optimality.

A number of heuristic models have been proposed by different researchers. The notable among them are due to Palmer (10), Gupta (3), Petrov (11), Cambell, Dudek and Smith (2), Nobeshima (7), Krone and Steiglitz (6), Gupta and Maykut (4). Palmer has suggested a quick method of scheduling. He assigns priority to items by determining a numerical slope index for each job and constructing a sequence in non-decreasing order of the slope index.

In Gupta's method (3), a numerical value is assigned to each job on the basis of a predefined function and the sequence is constructed in the ascending order of the values. Both Palmer and Gupta have used the analogy between scheduling and sorting problems, but Gupta claims that his method is superior to Palmer's algorithm.

Cambell, Dudek and Smith (2) have treated the problem by constructing (m-1) sub problems, where m is the number of machines. For each sub-problem, a sequence is obtained using Johnson's 2-machine optimum method. The sequence which yields the minimum make-span is finally selected.

Petrov's algorithm (11) is similar to Campbell, Dudek and Smith's algorithm (2) with one important difference that it does not use Johnson's algorithm.

Petrov's algorithm has been used in the present work for the generation of initial sequence. The details of this algorithm are presented in Appendix A.

Krone and Steiglitz (6) have proposed a heuristic programming approach with mean job completion time objective function. They fix a starting schedule by a pseudo-random algorithm and use a heuristic programme which computes a number of trial sequences. The algorithm terminates when local optimality of the current trial solution is verified by searching its neighbourhood exhaustively.

Page (8,9) has developed four heuristics based on sorting techniques, merging (M), pairing (P), individual exchanging (IE), and group exchanging (GE). The merging approach considers the list of jobs as a collection of strings of items. Initially the list is divided into n strings, each containing one job. Each successive pair of strings is then merged into a single ordered string containing two jobs. The ordering is based on the minimum make-span for the pair of jobs. Continuing in this manner, the number of jobs per string is increased while the number of strings is decreased until a single ordered string of n items remains. The pairing procedure is almost identical to that for merging, with the exception that once a string has been formed, adjacent

jobs will remain permanently adjacent in any larger strings developed.

The individual exchanging heuristic is based on the exchange method of sorting. Starting with a given job order, a test is performed to see whether each successive pair of adjacent jobs should remain as they are ordered or exchange positions. An exchange is made whenever the new sequence has a lower make-span. If an advantageous exchange is found on the first pass, a second pass is made. The process continues until a pass fails to yield a profitable exchange.

This is infact a very inefficient way of making the exchanges. Some of the exchanges can be dropped outrightly because they would worsen the sequence if made. Moreover, Page's procedure confines only to adjacent pairwise exchange and the higher order exchanges are not considered at all. In the present work an algorithm has been developed which takes care of these drawbacks.

Lastly, Page's Group Exchange Method is quite similar to individual exchange method except that groups of jobs are exchanged rather than individual jobs.

CHAPTER III

FOUNDATIONS OF PROPOSED METHODOLOGY

In Chapter II a review of the heuristic procedures, available in the literature for flowshop scheduling problems, was presented. It was pointed out that, in general, these heuristic procedures are computationally efficient but do not necessarily yield optimal sequences. The problem under consideration is to evolve a methodology which when used on an arbitrary sequence would improve upon the sequence which may further be used as an input sequence for further improvement. The procedure would have to be iterative in nature and would terminate when no further improvements are feasible considering a prespecified measure of effectiveness. In the present work makespan time has been used as the criterion of effectiveness of a sequence. The initial sequence, instead of being generated arbitrarily, is developed using one of the best heuristics available in the literature for flowshop scheduling problem.

The problem can now be formally stated as follows:

3.1 Statement of the Problem:

Given a sequence for a static flowshop scheduling problem which has been generated arbitrarily ... ...

or through the use of a heuristic, develop a methodology which exploits certain characteristics of this schedule for the generation of improved sequence considering makespan time as the criterion of effectiveness.

The above problem statement actually focuses on the ultimate goal, the statement would acquire the following structure if it is analysed in terms of how to achieve the goal.

Consider a static flowshop involving **n** jobs and m machines. Let t_{ij} represents the processing time of j-th job on i-th machine. Further, let

$$S = J(1)-J(2) \ldots\ldots - J(j) \ldots\ldots J(n)$$

represents a sequence with J(j) representing the job in position j of the sequence. Given a sequence S and processing time matrix $T = \left\{t_{ij}\right\}$, find the job positions $j^{\dagger}_{out}$ and $j^{\dagger\dagger}_{in}$ for the cyclic pairwise exchange in the given sequence such that the makespan time of the new sequence S^{*}, obtained after making the said exchange, is lower than that of previous sequence. The term cyclic pairwise exchange is explained in the next section.

Each exchange constitutes an iteration, if an iteration yields an improvement, the next iteration is undertaken. The process is continued until no further improvement is possible.

3.2 Cyclic Pairwise Exchange:

Consider a sequence S = J(1)-J(2)... J(n) for cyclic pairwise exchange which is to be effected to displace J(k) job to position i. The exchange under consideration can be specified as $\{J(k), i\}$. The procedure involves the removing of job J(k) and shifting the string of remaining jobs lying between positions i and k by one position towards k such that the i-th position is vacated and the k-th position gets occupied. In this process the job J(k) gets placed at position i. It should be noted that jobs beyond i-k range remain unaffected.

For the sake of illustration, consider a sequence S = 4-3-1-2-5. Let us say job J(4) = 2 is to be displaced to position i = 2, in other words the exchange $\{2,2\}$ is to be made. The new sequence after making the said exchange will be:

$$S^{*} = 4 - 2 - 3 - 1 - 5.$$

3.3 An Approach for the Development of Proposed Methodology:

In this section the basic framework which is used for the development of the proposed methodology is

---

$j^{\dagger}_{out}$ is the job position in the sequence from where the job should be taken out.

$j^{\dagger\dagger}_{in}$ is the job position in the sequence where the job $J(j_{out})$ has to come in.

presented. Throughout this work an operation is specified by task position in the sequence and the machine on which it is to be performed. For example, an operation represented as (i, j) connotes that the job in the j-th position of the sequence is to be performed on machine i.

As has been pointed out earlier, certain important characteristics of the given sequence which is assumed to be sub-optimal in character, need to be exploited for the identification of cyclic pairwise exchanges. Obviously, one such characteristic would be the idle time associated with some of the jobs and machines. In the process, a job may have to wait for want of availability of the machine and similarily a machine may have to wait for want of job. These waiting times are referred as idle times here. For the identification of such idle times, one alternative is to plot Gantt chart for the given sequence. However, to facilitate the computerization of the proposed methodology a matrix approach is adopted for obtaining the idle times. A matrix referred as Earliest Finish Gantt (EFG) matrix is developed first and is used for generating the idle times for the various operations in a matrix form.

3.3.1 Earliest Finish Gantt (EFG) Matrix:

This is an m x n matrix which represents the earliest completion times of various operations and is denoted by $G = \{g_{ij}\}$. The element g_{ij} indicates the earliest completion time of operation (i, j).

The following relationships can be used for the development of EFG matrix:

$$g_{ij} = \max [g_{i,j-1}, g_{i-1,j}] + t_{i,J(j)}$$

$$g_{0,j} = 0 \qquad j = 1,2,\ldots n$$

$$g_{i,0} = 0 \qquad i = 1,2,\ldots m$$

It should be noted that the element $g_{m,n}$ will represent the makespan time of the given sequence. The EFG matrix information can be presented in the form of a chart which will be referred as EFG chart. On this chart, corresponding to each operation there is a horizontal bar which is called Operation Block.

3.3.2 Idle Time Matrix:

The idle time matrix $I = \{I_{ij}\}$ can be generated by the following relationships:

$$I_{i,j} = g_{i-1,j} - g_{i,j-1}$$

$$I_{1,j} = 0 \qquad j = 1,2,\ldots, n$$

$$I_{i,1} = g_{i-1,1} \qquad i = 2,3,\ldots, m$$

An element of this matrix can take any value. Negative $I_{i,j}$ would imply that the job in the j-th position of the sequence S awaits the i-th machine for $|I_{i,j}|$ time. A positive $I_{i,j}$ signifies that the i-th machine waits for the job in the j-th position for this period. Obviously, $I_{i,j} = 0$ means that the job in position j is immediately taken up by i-th machine.

The concepts of Latest Finish Gantt (LFG) matrix, slack time (matrix) and critical path have been introduced to facilitate the development of the methodology.

3.3.3 Latest Finish Gantt (LFG) Matrix:

This matrix represents the latest completion times of the various operations and is denoted by $G' = \left\{ g'_{i,j} \right\}$, where $g'_{i,j}$ indicates the latest completion time of the operation (i, j). It may be noted that for a sequence, $g'_{i,j}$ will always be greater than or equal to $g_{i,j}$. An algorithm has been developed for the generation of LFG matrix.

The various steps of the algorithm are given below:

Step 1: For a given sequence S = J(1)-J(2),..., J(n) and machine ordering 1,2,..., m, obtain the EFG matrix $G = \left\{ g_{ij} \right\}$. Find out the makespan time $M_s = g_{m,n}$.

Step 2: Reverse the sequence as well as the machine ordering, new sequence will be $S_R = J(n)-J(n-1),\ldots, J(1)$ and the machine ordering m, m-1, ..., 2,1.

Step 3: Find EFG matrix for the new case. Let this matrix be $\bar{G} = \left\{ \bar{g}_{ij} \right\}$,

where $\bar{g}_{ij} = \max \left\{ \bar{g}_{i,\, j-1},\ \bar{g}_{i-1,j} \right\} + t_{m+1-i,n+1-j}$

$\bar{g}_{0,j} = 0 \qquad j = 1,2,\ldots, n$

$\bar{g}_{i,0} = 0 \qquad i = 1,2,\ldots, m$

Step 4: Obtain a matrix $\bar{\bar{G}} = \left\{ \bar{\bar{g}}_{il} \right\}$,

where, $\bar{\bar{g}}_{ij} = M_s - \bar{g}_{ij}$

Step 5: Reverse the matrix $\bar{\bar{G}}$ in such a manner that i-th row becomes (m+1-i) th row and j-th column becomes (n+1-j)-th column. Let this matrix be called $\bar{\bar{G}}'$. The $\bar{\bar{G}}'$ matrix gives the latest start time of various operations.

Step 6: Obtain the LFG matrix $G' = \left\{ g'_{i,j} \right\}$,

where $g'_{ij} = \bar{\bar{g}}'_{ij} + t_{i,J(j)}$

3.3.4 Slack Time:

Slack time is the maximum amount by which an operation can be delayed without affecting the makespan time. The slack time matrix $K = \left\{ k_{ij} \right\}$ can be obtained

by subtracting the EFG matrix from the LFG matrix. Thus

$$K = G' - G$$

It needs to be pointed out that the operations (m, n) and (1, 1) will always have zero slack time.

As would become obvious in Chapter IV, the slack time matrix becomes necessary for determining the j_{in} position in the given sequence for making the cyclic exchange. Instead of generating the slack matrix for the complete sequence, only the matrix for the sequence of size (n-1), obtained after removing job $J(j_{out})$ from the sequence, need to be generated.

3.3.5 Critical Path:

Critical path is the path obtained by joining those (m+n-1) operations each of which has zero slack time. Obviously, the makespan time is changed if any of the operations on the critical path is shifted. There can be more than one critical path corresponding to a given sequence.

A critical path is denoted by a string of (m+n-1) operations. For example, (3,4)-(2,4)-(2,3)-(2,2)-(1,2)-(1,1) can be a critical path for a 3-machine, 4-job problem.

The critical path(s) corresponding to a given sequence can conveniently be depicted on a grid diagram. In a grid diagram, the horizontal and vertical lines

denote the machines and jobs, respectively. The nodes represent the operations. In Fig. 3.1 a grid diagram for a 4-machine, 6-job problem is presented. Further,

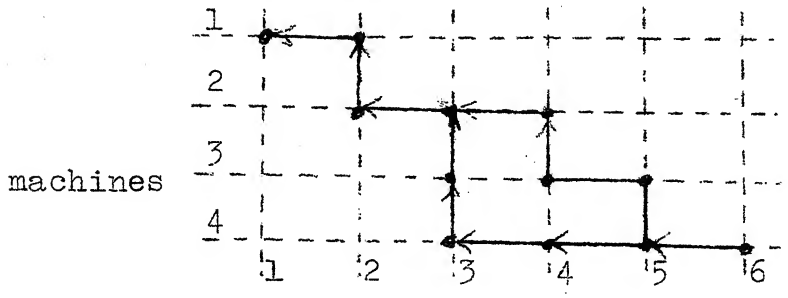


Fig. 3.1

the critical paths

(4,6)-(4,5)-(3,5)-(3,4)-(2,4)-(2,3)-(2,2)-(1,2)-(1,1)

and (4,6)-(4,5)-(4,4)-(4,3)-(3,3)-(2,3)-(2,2)-(1,2)-(1,1) are indicated on this diagram.

Following are some of the important properties of critical paths:

1. If an operation has zero slack time, at least one critical path will pass through it.

2. All critical paths start from operation (m,n) and end at operation (1,1).

3. If some operation (i,j) belongs to a critical path then the next operation on that path will either be (i-1,j) or (i,j-1) provided (i,j) is not (1,1).

4. There will not be any gaps between the operation blocks lying on a critical path.

5. If the slack time of operations (i-1, j) and (i, j-1) is zero, the idle time associated with the operation (i, j) will be zero.

6. If the operations (i-1, j) and (i, j) have zero slack then the operation (i, j) will have non-negative idle time.

7. If the operations (i, j-1) and (i, j) have zero slack then the operation (i, j) will have non-positive idle time.

Two important theorems along with their proofs are presented next.

Theorem 1: If an operation (i, j) has zero slack then at least one of the two operations, viz., (i-1, j) and (i, j-1) will have zero slack.

Proof: In the EFG matrix, the earliest finish time of an operation (i, j) is given by

$$g_{i,j} = \max \left\{ g_{i-1,j} , g_{i,j-1} \right\} + t_{i,J(j)}$$

The earliest start time for an operation (i, j) can be obtained by subtracting $t_{i,J(j)}$ from $g_{i,j}$. Thus the earliest start time is given by,

$$g_{i,j} - t_{i,J(j)} = \max \left\{ g_{i-1,j} , g_{i,j-1} \right\} .$$

The above relationship suggests that the earliest finish time of at least one of the two operations, viz., (i-1, j) and (i, j-1) would coincide with the earliest start time of operation (i, j). The operation whose earliest finish time coincides with the earliest start time of operation (i, j) can not be shifted relative to operation (i, j) which already has zero movability because of its zero slack. Therefore, at least one of the operations (i-1, j) and (i, j-1) would have zero slack.

Theorem 2: If an operation (i, j) has zero slack then at least one of the two operations, viz., (i+1, j) and (i, j+1) will have zero slack.

Proof: The movability of operation (i, j) is dictated by the operation (i+1, j) and (i, j+1) which are the only direct successors to operation (i, j). By contradiction, if both the operations (i+1, j) and (i, j+1) have positive slack then the operation (i, j) will have no operation restraining its movement towards the greater value on the time axis. Since the operation (i, j) has zero slack, it implies that at least one of the operations (i+1, j) and (i, j+1) will have zero slack.

The above theorems essentially imply that it is always possible to construct at least one critical path for a given sequence.

3.3.5.1 Algorithm for Critical Path Generation:

An algorithm has been developed for generation of critical paths corresponding to a given sequence. In order to save on computational effort the algorithm uses the information about Idle time matrix rather than slack time matrix. The various steps for the algorithm are:

Step 1: Consider the Idle time matrix $I = \{ I_{i,j} \}$ for the given sequence.

Step 2: Start from the operation (m,n). Set i = m, j = n and C = 0, where C identifies the critical path number.

Step 3: Set C = C + 1

Step 4: If $I_{i,j}$ is negative, go to Step 5
positive, go to Step 6
zero , go to Step 7

Step 5: Connect operations (i,j) and (i,j-1). Set j = j-1. Go to Step 8.

Step 6: Connect operations (i,j) and (i-1,j). Set i = i-1, Go to Step 8.

Step 7: Connect operation (i,j) with (i,j-1) and (i-1,j) separately. Set, i = i-1. Keep $\{(i,j-1); C\}$ in the memory.

Step 8: If i = 1 or j = 1, go to next Step. Otherwise go to Step 4.

Step 9: If i = 1, join (1,j) with (1,1) decreasing j by one in each step.

If j = 1, join (i,1) with (1,1) decreasing i by one in each step.

Step 10: If there is nothing left in the memory, STOP. Otherwise, take the first set $\{(i,j); C\}$ from the memory and remove it from there. Join operation (i,j+1) with (m,n) through critical path C, and go to Step 3.

3.4 An Example:

The various concepts introduced in this chapter are illustrated through an example.

Consider a 4-machine, 6-job flow shop with the following processing time information:

$$T = \begin{bmatrix} 5 & 10 & 6 & 6 & 8 & 9 \\ 16 & 3 & 9 & 4 & 5 & 7 \\ 12 & 4 & 8 & 7 & 11 & 6 \\ 3 & 7 & 5 & 13 & 3 & 8 \end{bmatrix}$$

In all, there will be 6! feasible sequences. Let us consider one of these feasible sequences. Let this sequence be S = 4-5-1-6-3-2. In this sequence J(4) = 6 indicates that job 6 takes 4th position in the

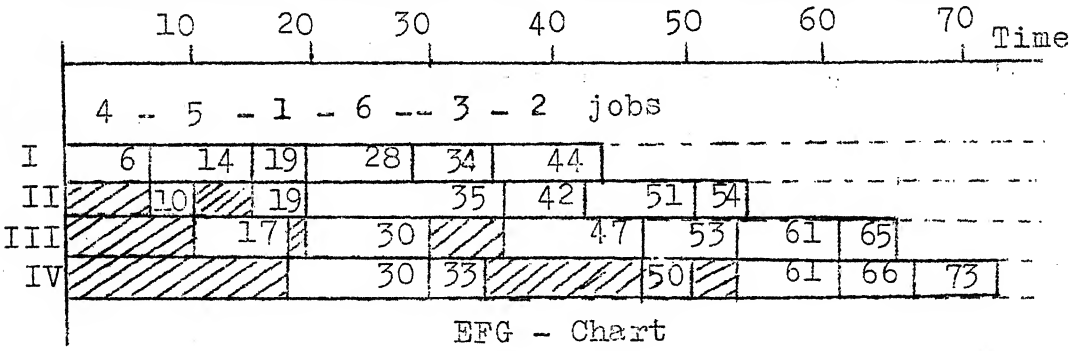
sequence and will be processed when the first 3 jobs, viz., 4-5-1 have been completed.

EFG Matrix and Chart:

For the above sequence and the machine ordering 1,2,3,4, the EFG matrix will be

$$G = \begin{bmatrix} 6 & 14 & 19 & 28 & 34 & 44 \\ 10 & 19 & 35 & 42 & 51 & 54 \\ 17 & 30 & 47 & 53 & 61 & 65 \\ 30 & 33 & 50 & 61 & 66 & 73 \end{bmatrix}$$

The corresponding EFG-chart is shown in Fig. 3.2. The shaded areas indicate the machine waiting times.



EFG - Chart
Fig. 3.2

Idle Time Matrix:

The idle time matrix will be as follows:

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 4 & 0 & -10 & -11 & -10 \\ 10 & 2 & 5 & -5 & -2 & -7 \\ 17 & 0 & 14 & 3 & 0 & -1 \end{bmatrix}$$

Here, $I_{2,4} = -10$ means that the operation (2,4) awaits machine 2 for 10 units.

Note that the various shaded areas in Fig. 3.2 correspond to the positive numbers in the I matrix.

LFG Matrix:

The algorithm given in Section 3.3.3 is applied for generating the LFG matrix.

Step 1: From the G-matrix, already obtained, $M_s = 73$.

Step 2: S_R = 2-3-6-1-5-4, and machine ordering: 4,3,2,1

Step 3:

$$\bar{G} = \begin{bmatrix} 7 & 12 & 20 & 23 & 26 & 39 \\ 11 & 20 & 26 & 38 & 49 & 56 \\ 14 & 29 & 36 & 54 & 59 & 63 \\ 24 & 35 & 45 & 59 & 67 & 73 \end{bmatrix}$$

Step 4:

$$\bar{\bar{G}} = \begin{bmatrix} 66 & 61 & 53 & 50 & 47 & 34 \\ 62 & 53 & 47 & 35 & 24 & 17 \\ 59 & 44 & 37 & 19 & 14 & 10 \\ 49 & 38 & 28 & 14 & 6 & 0 \end{bmatrix}$$

Step 5:

$$\bar{\bar{G}}' \quad \begin{bmatrix} 0 & 6 & 14 & 28 & 38 & 49 \\ 10 & 14 & 19 & 37 & 44 & 59 \\ 17 & 24 & 35 & 47 & 53 & 62 \\ 34 & 47 & 50 & 53 & 61 & 66 \end{bmatrix}$$

Step 6:

$$G' = \begin{bmatrix} 6 & 14 & 19 & 37 & 44 & 59 \\ 14 & 19 & 35 & 44 & 53 & 62 \\ 24 & 35 & 47 & 53 & 61 & 66 \\ 47 & 50 & 53 & 61 & 66 & 73 \end{bmatrix}$$

Slack Time Matrix:

Using the matrices G' and G, the slack time matrix K is obtained,

$$K = \begin{bmatrix} 0 & 0 & 0 & 9 & 10 & 15 \\ 4 & 0 & 0 & 2 & 2 & 8 \\ 7 & 5 & 0 & 0 & 0 & 1 \\ 17 & 17 & 3 & 0 & 0 & 0 \end{bmatrix}$$

Critical Path:

The various critical paths are constructed using the algorithm given in Section 3.3.5. The basic input is the I matrix which is reproduced below:

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 4 & 0 & -10 & -11 & -10 \\ 10 & 2 & 5 & -5 & -2 & -7 \\ 17 & 0 & 14 & 3 & 0 & -1 \end{bmatrix}$$

Critical path C = 1:

Starting from the operation (4,6) and using the I-matrix we get,

$I_{4,6} = -1$, so move horizontally to (4,5)

$I_{4,5} = 0$, move to (3,5) and keep $\{(4,4); 1\}$ in the memory

$I_{3,5} = -2$, move to (3,4)

$I_{3,4} = -5$, move to (3,3)

$I_{3,3} = +5$, move to (2,3)

$I_{2,3} = 0$, move to (1,3) and keep $\{(2,2); 1\}$ in the memory

Since i = 1, join (1,3) with (1,1).

Hence the critical path C = 1 is:

(4,6)-(4,5)-(3,5)-(3,4)-(3,3)-(2,3)-(1,3)-(1,2)-(1,1).

Look into the memory: We have $\{(4,4); 1\}$ and $\{(2,2); 1\}$. Let us take the first set first for the generation of another critical path and remove it from the memory.

Critical Path C = 2:

Take (4,6) to (4,5) portion of critical path C = 1. Start from (4,4),

$I_{4,4} = +3$, move to (3,4)

$I_{3,4} = -5$, move to (3,3)

$I_{3,3} = +5$, move to (2,3)

$I_{2,3} = 0$, move to (1,3) and keep $\{(2,2); 2\}$ in the memory. Since i = 1, join(1,3) with (1,1). This completes the identification of the second critical path.

Thus, the critical path C = 2 is:

(4,6)-(4,5)-(4,4)-(3,4)-(3,3)-(2,3)-(1,3)-(1,2)-(1,1)

Now the memory has $\{(2,2);\ 1\}$ and $\{(2,2);\ 2\}$ sets.
Selecting $\{(2,2),\ 1\}$ first, we generate next critical path.

Critical Path C = 3:

Portion of critical path C = 1 from (4,6) to (2,3) is:

(4,6)-(4,5)-(3,5)-(3,4)-(3,3)-(2,3)

Starting from (2,2).

$I_{2,2}$ = + 4, move to $I_{1,2}$

Since i = 1, join (1,2) with (1,1)

This completes the identification of 3rd path, viz.,

(4,6)-(4,5)-(3,5)-(3,4)-(3,3)-(2,3)-(2,2)-(1,2)-(1,1)

No more addition in the memory.

Taking the only set $\{(2,2),\ 2\}$ left in the memory.

Critical Path C = 4:

Portion of critical path C = 2 from (4,6) to (2,3) is:

(4,6)-(4,5)-(4,4)-(3,4)-(3,3)-(2,3)

Start from (2,2)

$I_{2,2}$ = + 4, move to $I_{1,2}$

Since i = 1, join (1,2) with (1,1).

Hence the critical path C = 4 is

(4,6)-(4,5)-(4,4)-(3,4)-(3,3)-(2,3)-(1,2)-(1,1)

Nothing left in the memory

Therefore, STOP.

CHAPTER IV

METHODOLOGY FOR IMPROVING THE SEQUENCE

4.1 Introduction:

In this chapter a methodology for improving a given sequence considering makespan time as the criterion of optimization is presented. The methodology draws upon the conceptual framework of slack matrix and critical path which were introduced in Chapter III. For improving upon a given sequence, the methodology uses cyclic pair wise exchanges for which j_{out} and j_{in} need to be identified. The identification of j_{in} and j_{out} has to be such that the exchange results in reduction of the makespan with respect to the initial sequence. The process of cyclic exchanges is carried out till no further improvement is possible. The methodologies for the identification of j_{out} and j_{in} are discussed separately and then the complete algorithm is presented.

4.2 Identification of j_{out} Position:

For the identification of j_{out} position the following heuristic rule is developed.

Rule: Only those jobs should be considered for removal for the cyclic exchange whose positions appear more than

once in at least one of the critical paths corresponding to the given sequence.

4.3 Identification of j_{in} Position:

A systematic methodology has been developed for the purposes of identifying the j_{in} position. The methodology utilizes the concept of 'unsatisfied' critical path which is discussed and explained with the help of an example.

4.3.1 Unsatisfied Critical Path:

A critical path is said to be unsatisfied by an exchange $\{J(k), i\}$ if:

1. The job positions i and k appear just once in critical path under consideration, and

2. The job positions i and k have same machine number in their respective operations of the critical path under consideration.

If any of the above stated considerations is violated, the critical path is said to have been satisfied by the exchange.

To illustrate the above definition an example is presented.

Let us consider a critical path which is represented as: (4,6)-(4,5)-(3,5)-(3,4),(3,3)-(3,2)-(3,1)-(2,1)-(1,1).

For this critical path it is observed that the job positions 2 and 4 appear only once and for both the job positions the associated machine is the same, i.e., machine number 3. Therefore the cyclic exchanges $\{J(4),2\}$ and $\{J(2), 4\}$ will 'unsatisfy' the given critical path. On the other hand the $\{J(3), 5\}$ is one of the exchanges that would satisfy this critical path.

Having introduced the concept of 'unsatisfied' critical path, a theorem is presented for the identification of plausible j_{in} positions.

Theorem: If a cyclic exchange $\{J(k), i\}$ results in some of the critical path(s) of a given sequence to remain 'unsatisfied', then the new sequence obtained by the said exchange will have makespan greater than or equal to that of the original sequence.

Proof: Since the exchange $\{J(k), i\}$ unsatisfies some of the critical paths, both the operations having job positions i and k and lying on these critical paths, will have the same machine number. Since there are no gaps between operations lying on a critical path, the operations lying between job positions i and k on the critical paths can only have non-positive idle times. Therefore, if the exchange $\{J(k), i\}$ is made, the time span contribution due to the operations lying between the two operations which took part in the exchange will

not decrease in any case, hence the makespan will not decrease. However, the above exchange may cause the other operations to arrange themselves in such a manner that some gaps get introduced on the critical path(s) under consideration. Such an exchange will increase the makespan time.

The above stated theorem is explained below with the help of an example.

Consider a critical path (4,6)-(3,6)-(3,5)-(3,4)-(3,3)-(3,2)-(2,2)-(1,2)-(1,1) which is depicted in a grid diagram as well as in EFG chart as shown in Fig. 4.1

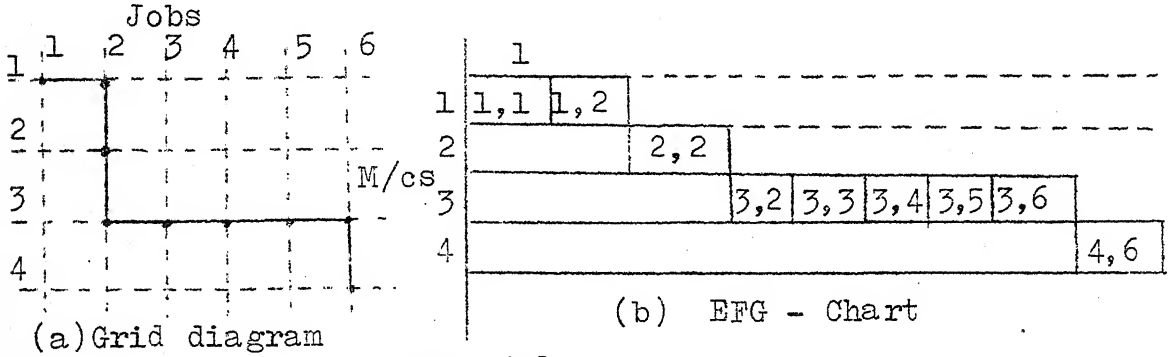


Fig. 4.1

$\{J(3),5\}$ is one of the cyclic exchanges that would unsatisfy the above mentioned critical path. If this exchange is made, the time span between operations (3,3) and (3,5) will not decrease, because there are no gaps between the two operations which could be reduced or vanished completely. However, some gaps can get introduced on the critical path due to reshuffling of

operation blocks. Hence the makespan will either increase or remain same.

The theorem discussed above gives rise to the following rule.

Rule: For improving a given sequence, only those cyclic pairwise exchanges should be considered which satisfy all the critical paths corresponding to the given sequence.

It should be noted that corresponding to an identified j_{out} there can be number of j_{in} candidates. The procedure to determine the set of jobs which can be removed, and their corresponding sets of j_{in} candidates is explained with the help of an example below.

Example:

Let us say following are the only critical paths corresponding to a given sequence

S = J(1)-J(2)-J(3)-J(4)-J(5)-J(6)

Critical path number C	Critical Path
1	(4,6)-(4,5)-(3,5)-(3,4)-(3,3)-(2,3)-(1,3)-(1,2)-(1,1
2	(4,6)-(4,5)-(4,4)-(3,4)-(3,3)-(2,3)-(1,3)-(1,2)-(1,1

Apply the rule mentioned in Section 4.2 to determine the set of jobs which can be removed from the given sequence. We get,

J(5) because job position 5 appears twice in critical path 1
J(4) because job position 4 appears twice in critical path 2
J(3) because job position 3 appears thrice in both critical paths

Next step is to determine the sets of j_{in} candidates for each of J(5), J(4) and J(3).

Consider each of these three jobs separately, say job J(5) is considered first. Now apply the rule mentioned in Section 4.3 to determine the two sets of j_{in} candidates with respect of the two given critical paths. We get,

Critical path number C	Set of j_{in} candidates
1	all positions
2	1, 2, 3, 4

The intersection of these two sets, i.e., 1,2,3,4 is the actual set of j_{in} candidates corresponding to J(5).

Hence, J(5) can be placed in position 1,2,3,4 of the given sequence S.

In the similar way, the sets of j_{in} candidates corresponding to jobs J(4) and J(3) can also be determined.

Once the set of j_{in} candidates is identified, the problem remains to select a specific j_{in} for exchange with

the already determined j_{out}.

4.3.2 Selection of j_{in}:

Once j_{out} is specified the information from slack and idle time matrices is utilized for the selection of a specific j_{in} out of the set of its candidates. The procedure involves the selection of that j_{in} which results in maximum reduction in the makespan time. This implies that one would have to generate the EFG matrix for calculating the makespan time after each exchange is made. This can be fairly time consuming. Therefore, procedure has been developed which determines the change in the makespan time due to an exchange $\{J(j_{out}), j_{in}\}$ without generating the EFG matrix. The development of the procedure is explained in the following section.

4.3.2.1 Procedure:

Given a sequence S = J(1)-J(2) ... J(n)

let j_{in} and j_{out} correspond to j and j', respectively. After removal of job J(j'), the sequence will have (n-1) jobs. Corresponding to this sequence of (n-1) jobs EFG, idle time and slack matrices are generated. Note that these matrices will be of size m x n-1.

If the removed job J(j') is placed at position j in the sequence of (n-1) jobs then what would be the increase in the makespan, is determined using a procedure discussed below.

At this stage we have the sequence of (n-1) jobs for which an EFG chart can be developed and further the makespan M_s' can be evaluated. Let us say that the removed job is to be placed in position j. This would displace the present job in position j along with its followers by one position. In Fig. 4.2 a portion of the EFG chart which corresponds to two jobs, viz., J(j-1) and J(j) is presented. The positive idle times are shown by hatched lines. One can utilize these idle times advantageously for accommodating the job to be inserted.

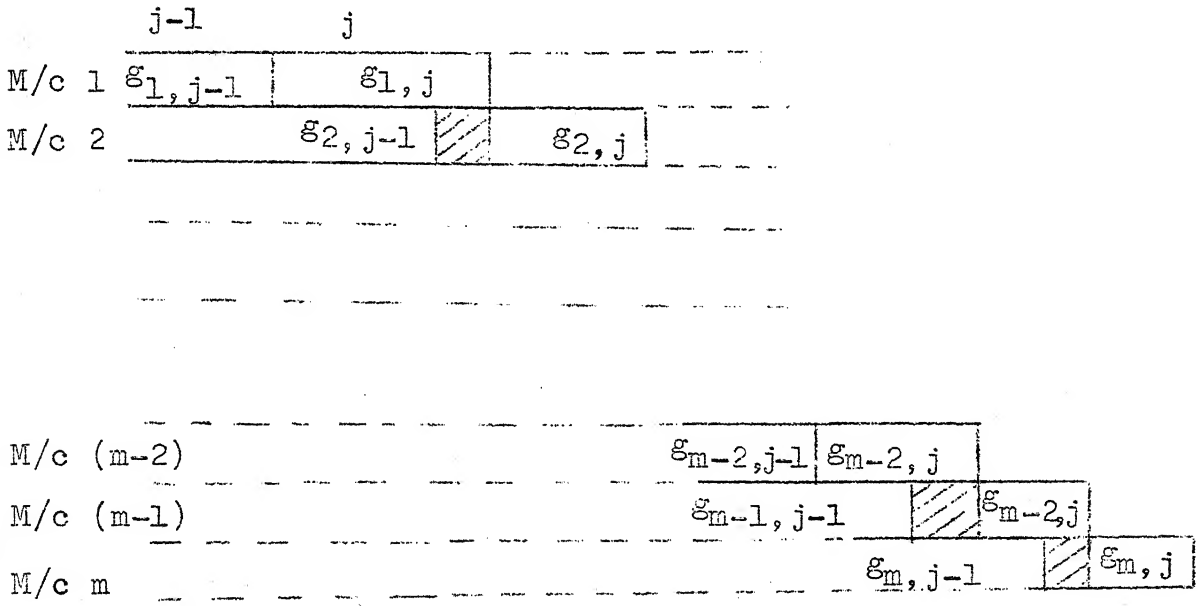


Fig. 4.2: EFG Chart

Further, the slack times associated with job in position j can also be utilized for this purpose. Therefore, the

total available gap at machine i for the insertion of the job J(j') is given by,

$$k_{ij} + \max\ [I_{ij}, 0]$$

However, there is a possibility that the actual available gap is some what less. The reduction in the gap will occur when the operation block of job J(j') corresponding to machine (i-1) is large enough to cover the operation block (i, j-1) as well as a portion of the total gap mentioned above. This kind of situation is depicted in Fig. 4.3.

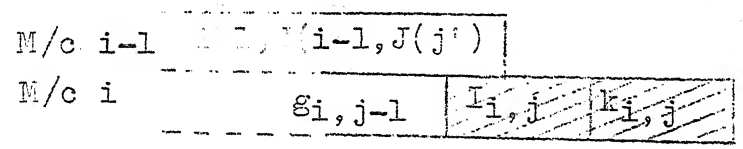


Fig. 4.3

Thus the net gap available for accommodating the operation of job J(j') on machine i, i.e., $t_{i,J(j')}$ is

$$P_i = k_{i,j} + \max\ [I_{i,j},0] - \max\ [(\tau_i - g_{i,j-1}),0]$$

where, $\tau_i = \tau_{i-1} + t_{i-1,J(j')}$ and $\tau_o = 0$

If $t_{i,J(j)}$ is greater than P_i, the available gap would have to be increased. Let the minimum shift which needs to be given to the jobs lying after position (j-1) to create this additional gap required corresponding to machine i be x_i. This implies that

$$x_i = t_{i,J(j')} - k_{i,j} - \max [I_{i,j}, 0] + \max [(\tau_i - g_{i,j-1}), 0]$$

So far we have been confining our attention towards accommodating i-th operation of J(j') on machine i only. Since this job is processed on all the machines, the minimum increment required in the makespan, M_s', will be $\underset{i}{\text{Max}}\ (x_i)$.

4.4 Algorithm:

The following are the steps of the final algorithm developed for improving upon a sequence.

Step 1: Obtain a sequence S = J(1)-J(2)... J(n) using any one of the known heuristics.

Step 2: Generate EFG-matrix $G = \{g_{i,j}\}$, idle time matrix $I = \{I_{i,j}\}$ for the sequence S. Determine makespan M_s.

Step 3: Construct critical path(s) using I matrix.

Step 4: Store the job positions which appear more than once in any of the critical paths that have been constructed in Step 3. Let these job positions are stored as set $\{V(k);\ k = 1,2,\ldots K\}$. K is total number of such positions.

Step 5: Set k = 1 and $M_s'' = \infty$

Step 6: Set v = V(k)

Step 7: Obtain a sequence S' = J'(1)-J'(2)...J'(n-1) after removing J(v) from sequence S.

Step 8: Generate EFG-matrix, I-matrix, K-matrix with respect to sequence S'. Determine makespan M_s'.

Step 9: Identify the set of j_{in} candidates in sequence S corresponding to job J(v). Let this set be denoted as $\{H(q);\ q = 1,2,\ldots L\}$, where L is total number of j_{in} candidates corresponding to J(v).

Step 10: Determine increment in makespan for each candidate H(q). Pick up the candidate which incurs the minimum increment and store it in j_{in}. Let the minimum increment be called MIN.

Step 11: If (MIN + M_s') is less than M_s'', go to Step 12. Otherwise go to Step 13.

Step 12: Set INCMT = MIN and j_{out} = v. Determine $M_s'' = M_s' + INCMT$.

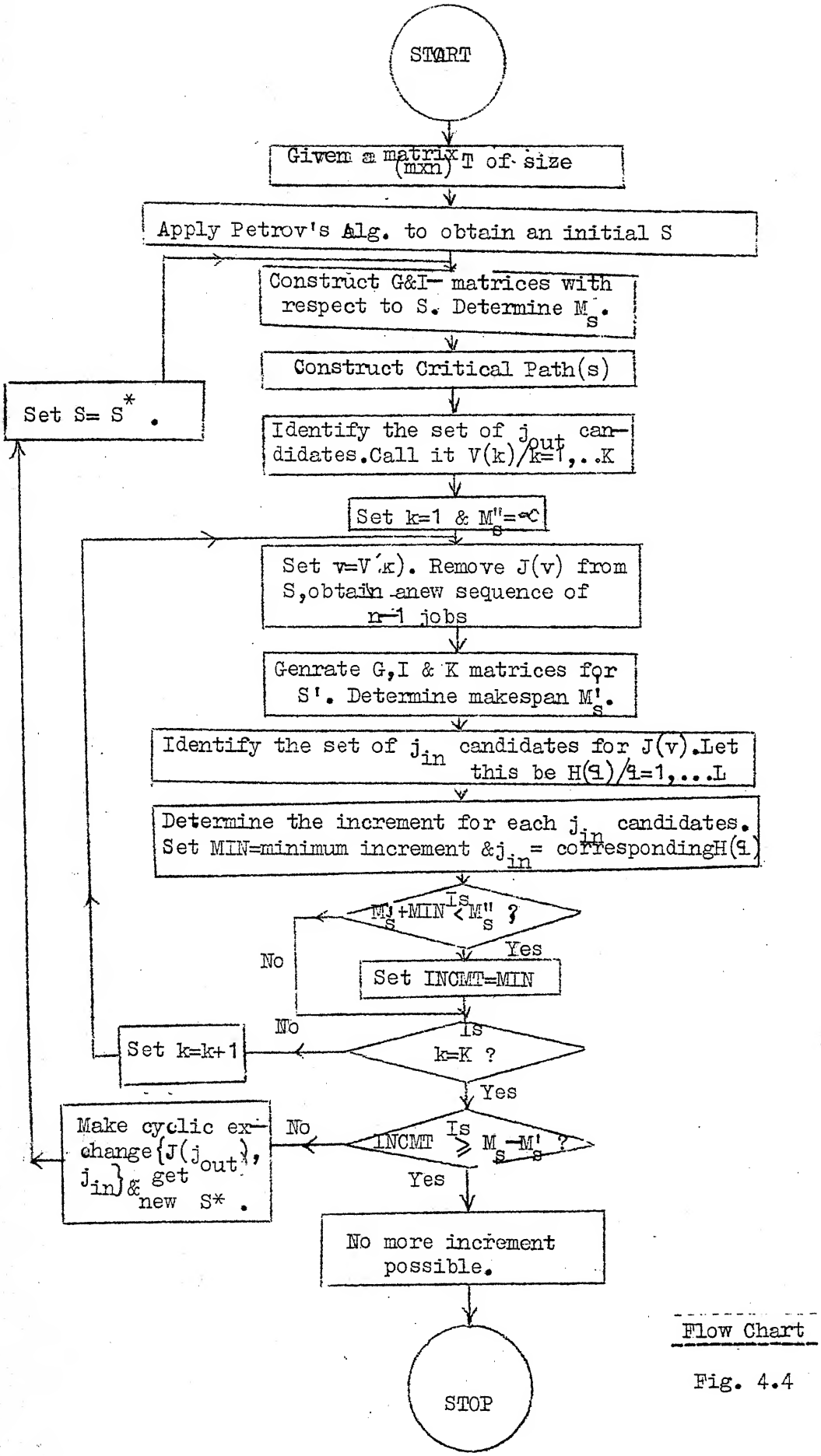
Step 13: If k = K, go to Step 15. Otherwise go to Step 14.

Step 14: Set k = k+1 and go to Step 6.

Step 15: If INCMT is greater than or equal $(M_s - M_s')$, go to Step 18. Otherwise go to Step 16.

Step 16: Make cyclic exchange $\{J(j_{out}), j_{in}\}$ on sequence S to obtain a new sequence S^*.

Step 17: Set S = S^* and go to Step 2.



Flow Chart

Fig. 4.4

Step 13: Print, 'No more improvement possible on the original sequence', Hence STOP.

A general logic flow diagram of the above stated algorithm is presented in Fig. 4.4.

4.5 Example:

To illustrate the various steps of the algorithm presented in the last section an example follows:

Given a processing time matrix

$$T = \begin{bmatrix} 10 & 12 & 1 & 13 & 6 & 5 & 19 & 5 & 7 \\ 15 & 7 & 8 & 19 & 5 & 14 & 16 & 19 & 18 \\ 7 & 10 & 11 & 16 & 17 & 11 & 7 & 12 & 13 \\ 21 & 8 & 12 & 3 & 1 & 13 & 9 & 11 & 15 \\ 18 & 3 & 14 & 14 & 11 & 3 & 11 & 15 & 13 \end{bmatrix}$$

for a 5-job, 9-machine flow shop problem.

Step 1: Using the Petrov's algorithm the following sequence is generated.

$$S = 3 - 5 - 1 - 8 - 9 - 4 - 7 - 6 - 2$$

Step 2:

$$G = \begin{bmatrix} 1 & 7 & 17 & 22 & 29 & 42 & 61 & 66 & 78 \\ 9 & 14 & 32 & 51 & 69 & 88 & 104 & 118 & 125 \\ 20 & 37 & 44 & 63 & 82 & 104 & 111 & 129 & 139 \\ 32 & 38 & 65 & 76 & 97 & 107 & 120 & 142 & 150 \\ 46 & 57 & 83 & 98 & 111 & 125 & 136 & 145 & 153 \end{bmatrix}$$

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 3 & -10 & -22 & -27 & -27 & -38 & -40 \\ 9 & -6 & -5 & 7 & 6 & 6 & 0 & 7 & -4 \\ 20 & 5 & 6 & -2 & 6 & 7 & 4 & 9 & -3 \\ 32 & -8 & 8 & -7 & -1 & -4 & -5 & 6 & 5 \end{bmatrix}$$

Makespan $M_s = 153$

Step 3: There will be only one critical path as given below:

(5,9)-(4,9)-(4,8)-(3,8)-(2,8)-(2,7)-(2,6)-(2,5)-
(2,4)-(2,3)-(1,3)-(1,2)-(1,1)

Step 4: The total number of candidates for j_{out} is $K = 3$ and they are $V(1) = 3$, $V(2) = 8$ and $V(3) = 9$.

Step 5: $k = 1$ and $M_s'' = \infty$

Step 6: $v = V(1) = 3$

Step 7: Remove $J(3) = 1$, we get

$S' = 3 - 5 - 8 - 9 - 4 - 7 - 6 - 2$

Step 8:

$$G = \begin{bmatrix} 1 & 7 & 12 & 19 & 32 & 51 & 56 & 68 \\ 9 & 14 & 33 & 51 & 70 & 86 & 100 & 107 \\ 20 & 37 & 49 & 64 & 86 & 93 & 111 & 121 \\ 32 & 38 & 60 & 79 & 89 & 102 & 124 & 132 \\ 46 & 57 & 75 & 92 & 106 & 117 & 127 & 135 \end{bmatrix}$$

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & -2 & -14 & -19 & -19 & -30 & -32 \\ 9 & -6 & -4 & 2 & 6 & 0 & 7 & -4 \\ 20 & 5 & 11 & 4 & 7 & 4 & 9 & -3 \\ 32 & -8 & 3 & 4 & -3 & -4 & 7 & 5 \end{bmatrix}$$

$$K = \begin{bmatrix} 0 & 2 & 2 & 14 & 19 & 19 & 30 & 39 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 14 & 14 & 14 & 12 & 7 & 7 & 0 & 3 \\ 19 & 27 & 16 & 12 & 13 & 9 & 0 & 0 \\ 19 & 19 & 16 & 12 & 12 & 12 & 5 & 0 \end{bmatrix}$$

Makespan $M_s' = 135$

Step 9: All the 9 job positions are the candidates for job $J(3) = 1$.

Step 10: Increments corresponding to each j_{in} candidate are,

Job position:	1	2	3	4	5	6	7	8	9
Increment:	24	17	18	15	15	15	15	21	36

Candidates 4,5,6,7 yield minimum increment. So any of these should be selected. Let us say candidate 4 is selected.

Therefore, $j_{in} = 4$ and $MIN = 15$.

Step 11: M_s'' is ∞ and $MIN + M_s' = 150$. Therefore go to Step 12.

Step 12: $INCMT = MIN = 15$, $j_{out} = v = 3$ and

$M_s'' = M_s' + INCMT = 150$

Step 13: k is not equal to K at this step, therefore go to Step 14.

Step 14: Set $k = 1+1 = 2$ and go to Step 6.

Cycle, from Step 6 to Step 14, is repeated for next j_{out} candidate which is V(2) = 8.

This yields INCMT = 10 j_{in} = 6 and $M_s^{'}$ = 139.

For the last candidate V(3) = 9 the steps from 6 to 11 are repeated. At Step 11 (MIN + $M_s^{'}$) happens to be greater than $M_s^{'}$, therefore it goes to Step 13.

Step 13: k = 3 and K = 3, go to Step 15.

Step 15: $M_s - M_s^{'}$ = 153 - 139 = 14

INCMT = 10

INCMT is less than ($M_s - M_s^{'}$), therefore go to Step 16.

Step 16: Make cyclic exchange $\{J(8), 6\}$ on sequence S = 3-5-1-8-9-4-7-6-2. The new sequence will be S* = 3-5-1-8-9-6-4-7-2.

Step 17: Set S = S*= 3-5-1-8-9-6-4-7-2 and go to Step 2.

This completes one iteration of the algorithm yielding a reduction of 4 units in the makespan.

In the second iteration a further reduction of 3 units is realized.

After the third iteration it stops without any more reduction.

So, finally we get the following sequence as the output of proposed heuristic.

S = 3-5-8-1-9-6-4-7-2

Makespan = 146

Total reduction = 153 - 146 = 7

CPU time = 0.15 seconds.

CHAPTER V

PERFORMANCE EVALUATION AND CONCLUSIONS

5.1 Performance Evaluation:

In this section, the results of experimental investigations carried out to evaluate the performance of the proposed heuristic procedure are presented. The evaluation is primarily done in terms of the number of times the heuristic succeeds in improving the initial sequence and the amount of reduction in the makespan time.

The experimental investigation was divided into two parts. Firstly, an attempt was made to solve a standard problem available in the literature with known optimal solution. The 3-machine, 10-job flowshop scheduling problem given in the book by Conway et.al.*, was solved using the proposed heuristic. Conway, et al., have claimed this to be the hardest flowshop scheduling problem available in the literature for solving through the use of heuristic procedures. The objective was to see as to how far the proposed heuristic brings the initial suboptimal sequence close to the optimal solution. The initial suboptimal sequence was generated using Petrov's algorithm.

* Conway, R.W., Maxwell, W.L., and Miller, L.W., Theory of Scheduling, Addison-Wesley, Reading, Mass., 1967, p. 94.

The second part of the investigation involved the testing of the proposed heuristic on a set of randomly generated problems of varied sizes. In all 13 problem sizes varying from 3-machine, 4-job to 7-machine, 10-job were considered. Each problem size was repeated five times. The processing times on the machines for various jobs were obtained using random number generators. Again, Petrov's algorithm was used for the generation of initial sequence.

5.1.1 Evaluation of the Standard Problem:

The processing time matrix of the standard problem stated earlier is given below:

$$T = \begin{bmatrix} 1 & 5 & 7 & 8 & 3 & 7 & 9 & 8 & 6 & 3 \\ 2 & 9 & 6 & 9 & 2 & 10 & 7 & 9 & 1 & 1 \\ 9 & 7 & 8 & 9 & 3 & 4 & 7 & 4 & 3 & 1 \end{bmatrix}$$

The optimum makespan time for this problem has been reported as 66. The Petrov's heuristic gives the following sequence.

S = 1-5-3-2-6-7-4-8-9-10 with a makespan of 69.

Using Petrov's solution as the initial sequence, the proposed heuristic in the first iteration gave a makespan value of 66. The corresponding sequence was 1-2-5-3-6-7-4-8-9-10.

Thus for the 'hardest problem' the proposed heuristic succeeded in arriving at the optimal sequence in just one iteration. The author has not been in a position to extensively test the ability of the proposed heuristic to generate optimal sequence for want of large-sized standard problems with known optimal solutions.

5.1.2 Evaluation on Randomly Generated Problems:

The results obtained on average makespan reduction for the randomly generated problems of various sizes are presented in Table 5.1. It was observed that the proposed heuristic improved the initial sequence for almost 70 per cent of the problems and the average improvement was 4 per cent. Further, it was noted that the tendency to improve upon the initial solution increased with the increase in the problem size.

In Table 5.1, the average computational requirements of the proposed heuristic for various problem sizes is also presented. The problems were solved on DEC 1090 computer system and for each problem size a set of 5 problems were considered. It is observed that the computational requirements for the proposed heuristic are reasonable even for problems involving 7 machines and 10 jobs.

5.2 Conclusions:

In this thesis, we have presented a heuristic methodology for improving upon an initial sequence for a static flowshop problem. The objective has been to exploit the inherent characteristics of the initial suboptimal sequence. The concepts of idle time, slack time, critical path and cyclic pairwise exchange have been utilized advantageously for the development of the proposed heuristic methodology. The experimental investigations have indicated that the heuristic algorithm improves the initial solution for most of the problems. The tendency to yield improved sequence increases with an increase in problem size. Further, it is observed that the methodology is computationally efficient.

Table 5.1

Sl. No.	Problem Size	Average reduction in makespan time, percent	Average CPU time (Seconds)
1.	3x4	0.08	0.05
2.	3x6	2.43	0.08
3.	3x8	2.90	0.09
4.	4x5	1.86	0.07
5.	4x6	3.30	0.08
6.	4x7	2.83	0.09
7.	5x7	4.96	0.11
8.	5x8	4.02	0.12
9.	5x9	5.86	0.15
10.	6x4	4.17	0.08
11.	6x9	5.31	0.20
12.	7x9	7.12	0.21
13.	7x10	6.19	0.25

REFERENCES

1. Bakshi, M.S. and S.R. Arora, 'The Machine Sequencing Problem', Management Science, Vol. 16, No. 4, 1969, pp. B 247-263.

2. Cambell, H.G., Richard A. Dudek and Milton L.Smith, 'A Heuristic Algorithm for the n-job m-machine Sequencing Problem, Management Science, Vol. 16, No. 2, 1970, pp. B 630-637.

3. Gupta, J.N.D., 'A Functional Heuristic Algorithm for the Flow-shop Scheduling Problems', Operational Research Quarterly, Vol. 22, No. 1, 1971, pp.39-48.

4. Gupta, J.N.D., and Albert R. Maykut, 'Heuristic Algorithms for Scheduling n-jobs in a Flowshop', Journal of Operations Research Society of Japan, Vol. 16, No. 3, 1973, pp. 131-150.

5. Johnson, S.M., 'Optimal Two and Three Stage Production Schedules with Set-up Times Included', Naval Research Logistic Quarterly, Vol. 1, No. 1, 1954, pp. 61-68.

6. Krone, M.J. and Kenneth Steiglitz, 'Heuristic Programming Solution of a Flow-Shop Scheduling Problem', Operations Research, Vol. 22, No.3,1974, pp. 621-628.

7. Nobeshima, I., 'The Order of n items Processed on m Machines', Journal of Operations Research Society of Japan, Vol. 16, No.3, 1973, pp. 163-185.

8. Page, E.S., 'An Approach to Scheduling Jobs on Machines', Journal of the Royal Statistical Society, Series B, Vol. 23, pp. 484-492.

9. Page, E.S., 'On the Scheduling of Jobs by Computer', The Computer Journal, Vol. 5, pp. 214-221.

10. Palmer, D.S., 'Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining a Near Optimum,'Operational Research Quarterly, Vol. 23, No. 3, 1972, pp. 323-331.

11. Petrov, V.A., Text Book, Flow Line Group Production Planning, London, Business Publications, 1968.

APPENDIX A

PETROV'S ALGORITHM

Given a processing time matrix $T = \{t_{i,j}\}$ for m machine, n job flowshop problem.

Define the corresponding 2-machine, n-job problem:

$A_j = \sum_{i=1}^{m_1} t_{i,j}$ = the processing time for the j-th job on machine 1.

$B_j = \sum_{i=m_2}^{m} t_{i,j}$ = the processing time for the j-th job on machine 2.

If m is an even number then,

$m_1 = \frac{m}{2}$ and $m_2 = m_1 + 1$

If m is an odd number then

$m_1 = \frac{m+1}{2}$ and $m_2 = m_1$

The following steps are applied to the above mentioned 2-machine problem:

Step 1: Determine a column matrix $\{C_j\}$ where $C_j = B_j - A_i$

Step 2: Apply rule No. 1:

Arrange all the jobs, having $C_j \geq 0$, in increasing value of C_j, and the jobs, having $C_j < 0$, in decreasing value of C_j. Call the sequence, hence generated, S_1.

Step 3: Apply Rule No. II:

Arrange all the jobs in decreasing value of C_j generating a sequence S_2.

Step 4: Select, from S_1 and S_2, the one which gives minimum makespan.

In addition to this, if there are some zero elements in the T matrix then two more sequences are generated as follows:

Define,

$$A'_j = \frac{\sum_{i=1}^{m_1} t_{i,j}}{N_j}$$

$$B'_j = \frac{\sum_{i=m_2}^{m} t_{i,j}}{N'_j}$$

where, N_j is the number of machines out of first m_1 machines having positive processing times for job j and, N'_j is the number of machines out of last $(n - m_2+1)$ machines having positive processing times for job j.

Apply Rule No. 1 and 2 mentioned above to A'_j and B'_j to generate two more sequences S_3 and S_4.

Select the best out of S_1, S_2, S_3 and S_4.